MAPPING XML SCHEMA COMPONENTS TO QUALIFIED JAVA COMPONENTS

TECHNICAL FIELD

5

[0001] The invention disclosed generally relates to computers and computer software. More specifically, the invention is generally related to program language mapping and conversion tools.

BACKGROUND OF THE INVENTION

10 [0002] One of the major challenges in computer operations is how to make different systems exchange information in an efficient and cost-effective manner. A key aspect to this challenge is the need for a means to enable applications using different languages to exchange information. In the past, when systems were more isolated, this challenge could be met with more static

15 solutions. These were labor intensive, frequently requiring extensive manual effort in mapping the languages, formats and attributes of different database and network systems to each other. If changes were made to one or more of the databases or networking platforms (something unavoidable for most businesses), then further changes were required in the integration code. In

20 addition to the time and expense required by these approaches, they were also prone to manual errors and loss of features due to the inevitable comprises forced by the complexities of integration.

[0003] As the demand for information sharing has grown, so has the need for more automated solutions to the exchange of information between systems.

25 The development of a feature rich and robust language like XML (Extensible Markup Language) is facilitating the cross-platform and application exchange of information much more than its predecessor HTML (Hyper Text Markup Language) or proprietary languages did. However, the efforts to date at automating the mapping of information between XML and object-oriented

languages like Java remain constrained by the inherent differences in their component models. Following past approaches, the attempts so far have focused on a simple one to one component mapping.

[0004] While these approaches may be adequate in simpler configurations like those involving a single namespace, they create a range of potentially unworkable problems in more complex systems. For example, mapping the more robust XML schemas to other object (e.g., Java) classes can lead to component name collisions. (XML schemas define shared markup vocabularies, the structure of XML documents which use those vocabularies, and provide hooks to associate semantics with them.) Further, component naming restrictions for schemas differ from those imposed by the Java standard. Thus, schema constructs may not have explicit naming mechanisms that would allow for simple mapping to Java component names, and complex information regarding the schema is needed to generate Java component names for such schema constructs. Moreover, parts of schemas change over time, requiring regeneration of the Java components from schema components. Across these multiple schema to Java component conversions, the unchanged schema components should ideally map to the previously designated Java component names. If not, significant impacts on the system as a whole may occur when the Java component names are changed, including the need to rework of other components dependent on the Java components.

[0005] In considering these problems, we have identified a number of issues associated with the consumption and transformation of XML schemas into other proprietary and object structures like Java classes. Several of the key challenges identified include:

[0006] 1) Uniquely identifying XML elements and types within the set of all unique legal XML schemas;

[0007] 2) Uniquely naming XML elements and types, so that the names adhere to Java naming standards (found, e.g., in the Java language

specification) for packages, classes, methods and variables, even through successive consumption efforts on modified XML schemas; and

[0008] 3) Mapping XML elements and types to reusable proprietary definitions for modular processing.

5   [0009] While other applications have the ability to consume XML schemas to create Java classes, they have not addressed how to handle more complex conversions involving these issues of code regeneration, unique name handling, adherence to naming standards, and multiple schema namespaces. Thus, there is a need for a better way to carry out the conversion of XML

10   schema to Java and other object definitions.

SUMMARY

[0010] The present invention provides a method, apparatus, and computer instructions for mapping and labeling XML schema elements and types. In an exemplary embodiment, each XML schema element and type is uniquely

15   labeled, using distinguishing parameters such as namespace to create name parts like a Hashing code and suffixes in order to achieve unique mapping. The parameters selected are predetermined in order to achieve naming that remains distinct across successive consumptions of a schema. The selection disclosed accomplishes unique labeling, while also permitting one to still

20   adhere to the strict naming standards of the target language (e.g., Java). As a result, schema components belonging to multiple namespaces can be mapped to Java components belonging to a single package.

BRIEF DESCRIPTION OF THE DRAWINGS

[0011] While the invention is defined by the appended claims, as an aid to

25   understanding it, together with certain of its objectives and advantages, the following detailed description and drawings are provided of an illustrative, presently preferred embodiment thereof, of which:

[0012] FIG. 1 is a block diagram of an information system consistent with the invention.

[0013] FIG. 2 is a block diagram illustrating a software program for generating a business object definition from an XML schema in accordance with a first embodiment of the invention.

[0014] FIG. 3 is a block diagram illustrating an application program environment for the first embodiment.

[0015] FIG. 4 is a flow chart of an illustrative conversion operation according to the first embodiment of the invention.

[0016] FIG. 5 is a table illustrating mapping of XML schema components to Java components in accordance with the first embodiment of the invention.

[0017] FIG. 6 is a table illustrating Java component naming generated from an exemplary XML schema snippet in accordance with the first embodiment of the invention.

DESCRIPTION OF THE PREFERRED EMBODIMENT

[0018] In a preferred embodiment of the invention, a conversion engine is provided that includes an algorithm for operating on one or more sets of XML schemas to generate proprietary data definitions that uniquely model the XML schemas. In addition to the sense given by the XML Schema recommendations of the W3C Consortium to "unique," by "unique mapping" we mean that (a) each XML schema element or type is mapped to a component of the conversion language (i.e., object language into which XML documents will be converted based on the mapping definition), and (b) each XML schema element or type is uniquely identified within the set of all unique (preferably legal) XML schema. By "unique naming," we mean that unique XML schema element or type (a) generate unique component names in the conversion language (e.g., Java) that substantially (preferably fully) adhere to the conversion language naming standards, and (b) generate the same unique name across multiple schema conversions. The resulting definitions for each schema are objects, and in the case of the commonly preferred conversion language, Java, are essentially Java objects. As such, each definition generated should adhere to the Java

naming standards. The mapped components should also remain distinct across multiple consumptions or transformations of the XML schemas.

[0019] This embodiment may be advantageously implemented as part of complex systems like business integration (BI) systems. One example of a BI system is WebSphere Business Integration Servers and programs, available from International Business Machines (IBM). A convenient way to implement this preferred embodiment is to include it as part of an automated object discovery agent, a tool for both performing automated object discovery and generating business object definitions.

[0020] An object discovery agent according to this preferred embodiment consumes each XML schema and creates business object definitions (e.g., Java objects). These objects can be used as containers for XML instances that adhere to the XML schema. In doing so, the schema conversion algorithm preferably satisfies each of challenges identified above, i.e., code regeneration, unique name handling, adherence to naming standards, and multiple schema namespaces.

[0021] When converting XML schemas to Java components, one preferred methodology includes the following function goals:

[0022] 1) Adhere to the character limitations as defined by the Java standard for various Java components (e.g., While XML schema element names can contain a period character ('.'), mapped Java variable names cannot).

[0023] 2) Adhere to common application restrictions with respect to component naming.

[0024] 3) Uniquely and distinctly map all schema components to Java components with respect to name.

[0025] 4) Distinctly name Java components so that successive conversions of XML schema to Java yield the same named Java components.

[0026] 5) Provide user customizability for further Java component name distinctions.

[0027] With reference now to the drawings and in particular FIG. 1, a pictorial representation of an information processing system in which the present invention may be implemented is depicted in accordance with certain presently preferred embodiments of the invention. In general, the routines which are executed when implementing these embodiments, whether implemented as part of an operating system or a specific application, component, program, object, module or sequence of instructions, will be referred to herein as computer programs, or simply programs. The computer programs typically comprise one or more instructions that are resident at various times in various memory and storage devices in an information processing or handling system such as a computer, and that, when read and executed by one or more processors, cause that system to perform the steps necessary to execute steps or elements embodying the various aspects of the invention.

[0028] A particular information handling or processing system for implementing the present embodiments is described with reference to FIG. 1. However, those skilled in the art will appreciate that embodiments may be practiced with any variety of computer system configurations including hand-held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, minicomputers, mainframe computers and the like. The embodiment may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

[0029] In addition, various programs and devices described here may be identified based upon the application for which they are implemented in a specific embodiment of the invention. However, it should be appreciated that any particular program or device nomenclature that follows is used merely for

convenience, and the invention is not limited to use solely in any specific application identified and/or implied by such nomenclature.

[0030] Referring now to FIG. 1, a computer system 110 consistent with the invention is shown. For purposes of the invention, computer system 110 may represent any type of computer, information processing system or other programmable electronic device, including a client computer, a server computer, a portable computer, an embedded controller, a personal digital assistant, etc. The computer system 110 may be a standalone device or networked into a larger system. In one embodiment, the computer system 110 is an eServer iSeries OS/400 computer available from International Business Machines of Armonk, N.Y.

[0031] The computer system 110 could include a number of operators and peripheral systems as shown, for example, by a mass storage interface 140 operably connected to a direct access storage device 142 via high speed bus interface 141, by a video interface 143 operably connected to a display 145, and by a network interface 146 operably connected to a plurality of networked devices 148 ... 149. The display 145 may be any video output device for outputting a user interface. The networked devices 148 - 149 could be desktop or PC-based computers, workstations, network terminals, or other networked information handling systems, connected by any one of a variety of networking systems including a local area network (LAN) 147, personal area network (PAN), or wide area network (WAN).

[0032] Computer system 110 is shown with a system environment that includes at least one processor 120, which obtains instructions or operation codes (also known as opcodes) and data via a bus 115 from a main memory 130. The processor 120 could be any processor adapted to support the debugging methods, apparatus and article of manufacture of the invention. In particular, the computer processor 120 is selected to support monitoring of memory accesses according to user-issued commands. Illustratively, the

processor is a PowerPC available from International Business Machines of Armonk, N.Y.

[0033] The main memory 130 could be one or a combination of memory devices, including random access memory, nonvolatile or backup memory (e.g., programmable or flash memories, read-only memories, etc.). In addition, memory 130 may be considered to include memory physically located elsewhere in a computer system 110, for example, any storage capacity used as virtual memory or stored on a mass storage device or on another computer coupled to the computer system 110 via bus 115.

[0034] The main memory 130 includes an operating system 131, a conversion program 132, a framework/connector module 134, and other programs 135. These other programs 135 could include a programming analysis and transformation tool. The conversion and connector programs are generally of the type of adapters or tools used to facilitate information exchanges between differing applications, such as enterprise applications using XML documents and web applications using Java objects. These are generally implemented in software, but can be implemented in a combination of hardware (firmware) and software. In an alternate embodiment, the adapter tool could include other mapping or override features, configured to interface with other programs or a user via a GUI (graphical user interface) at terminal 145. Although illustrated as integral programs, one or more of the foregoing may exist separately in the computer system 110, and may include additional components not described. Processor 120 implements the processes illustrated using computer implemented instructions, which may be located in a memory such as, for example, main memory 130, memory 142, or in one or more peripheral devices 148-149.

[0035] Turning now to FIG. 2, the major elements of a preferred component for converting XML schemas to XML—Java object definitions are illustrated. These program elements include a conversion engine 205, an automated

mapping and naming program. This program includes a mapping element, for uniquely mapping XML schema components to Java components, and a naming element, for applying a methodology that uniquely names each Java component. Specification 206 provides the rules used for mapping and naming for a given target technology, the one highlighted below being XML. However, one skilled in the art will readily appreciate how to apply these teachings to future successor or substitute languages (including those substantially similar in major functionality) to XML. In such a case, multiple specifications may also be loaded, although a single specification would likely be used in a typical adapter, operating to generate object definitions for converting between a target technology like XML and a single conversion language like Java.

[0036] The conversion engine 205 also receives as input the data structure of the conversion language. As Java is increasingly used in many business applications, it may well be the preferred application data structure 203 for discovery and use by many conversion engines 205. In the illustrated embodiment of an integration system, the typical output of conversion engine 205 will be a business object definition for use in converting XML and Java objects exchanged via the integration system.

[0037] FIG. 3 is a block diagram illustrating one such integration system architecture. In this case, the conversion engine 205 is part of an automated object discovery agent (ODA) 311 that functions to provide a business object definition to a runtime framework/connector component or adapter 310. The connector may be an application specific component, functioning to provide bi-directional connectivity to the application API or technology interface. The framework can function to provide a common runtime platform for implementing specific connectors, to provide broker interaction, logging and QoS services, and to assure uniform behavior and administration across a set of adapters. The ODA receives (e.g., discovers by an automated agent) XML schemas from a target application 305, which in the illustrated case is a DBMS

(data base management system) retrieving and storing records from data store 301 and exchanging information via connector 310. A business integration server 315 provides process integration services for the various enterprise applications and adapters, including different applications as illustrated by
5   application 320.

[0038] The operation of the preferred embodiment may be understood by reference to FIGS. 4 through 6. When a schema is first consumed, or when its modification requires a further conversion, ODA 311 is called up to perform the necessary business object definition generation (step 410). This can be
10   done in response to manual (user) input, or alternatively by an automated call in response to schema changes in Application A. In typical operation, the next step will be to load the new or modified schema, and apply the conversion algorithm (step 430). However, a user may also, optionally, be prompted to load new technology specifications or otherwise change the parameters used by
15   the ODA/conversion engine (steps 420, 425). This includes the ability to define alternative component mapping and naming conventions, as well as to confirm that uniqueness requirements are satisfied by changes in the specification.

[0039] Once loaded, the ODA/conversion engine 311 proceeds to apply the mapping and naming criteria, and generate a business object definition (step
20   440). FIG. 5 is a table illustrating a preferred mapping between XML schema components and Java components. FIG. 6 is a table illustrating an exemplary naming convention for an XML schema to Java component conversion. The resultant business object definition (in this case, preferably a Java object) is forwarded to the connector 310 for use in runtime object conversion. If another
25   schema is ready to be operated on, the process is repeated (steps 450, 460).

[0040] When naming Java components, the following steps are preferably taken:

[0041] 1) Users have the option of specifying a string (to group the generated components) that will be added as a prefix to the set of generated Java component names.

[0042] 2) For all schema components names, all Java unsupported characters are replaced within the name with an underscore character '_' to generate the Java component name.

[0043] 3) If the generated names exceed common application character length limitations, a hash code is generated based on the original schema component QName, the generated name is truncated and the hash code appended so that the length adheres to the character length limitations. A preferred form of hashing is generating an eleven character code based on any convenient hashing function, with ten characters used as the hash code itself and the first character representing a positive or negative (e.g., P or N). If the hash function returns integer values less than ten characters in length, the integer is padded (e.g., with leading zeroes).

[0044] The following is an exemplary naming convention for use in connection with the specified XML components:

[0045] 1) Elements (local and global): Map element names to member variable names. If two elements or an element and an attribute have the same name within a type definition in the schema, unique member variable names must be generated in the class definition corresponding to the schema type. By the Java naming standard, member variables in a class must be uniquely named. The following two step process uniquely names the member variables: (a) Generate a hash code based on the QName of the element and the string "Elem"; generate the member variable name using the element name suffixed with the hash code; (b) if name collisions exist after step 1, append a sequence number to the name generated in step a.

[0046] 2) Simple Types (named and anonymous): Map simple types to primitive Java types.

[0047] 3) Complex Types (named and anonymous): (a) Named complex types—Map complex type names to Java class names. Generate a hash code based on the QName of the complex type. The Java class name will be the complex type name suffixed with the generated hash code. (b) Anonymous

5 complex types—Create a string comprised of the concatenation of all the names of the components in the hierarchy for this anonymous complex type starting from the schema element including the target namespace of the schema if present. Generate a hash code using the string above. The name is generated by appending the hash code to the name of the immediate parent

10 component of the anonymous complex type.

[0048] 4) Attributes (local and global): Map attribute names to member variable names. If two attributes or an attribute and element have the same name within a type definition in the schema, unique member variable names must be generated in the class definition corresponding to the schema type.

15 By the Java standard, member variables in a class must be uniquely named. The following two step process uniquely names the member variables: (a) Generate a hash code based on the QName of the attribute and the string "Attr"; generate the member variable name using the attribute name suffixed with the hash code; (b) if name collisions exist after step 1, append a sequence

20 number to the name generated in step a.

[0049] 5) Groups (global): Same as named complex types.

[0050] 6) Attribute Groups (global): Same as attributes.

[0051] This may be further illustrated by consideration of the following XML schema snippet (Table 1), in connection with the table of FIG. 6:

25

TABLE I

Example: XML Schema Snippet
<xsd:schema targetNamespace="typesNS" xmins="typesNS' xmins:xsd=http://www.w3.org/2001/XMLSchema>

```
<xsd:complexType name="Product">
    <xsd:sequence>
        <xsd:element name="ID" type="xsd:string"/>
        <xsd:element name="Name" type="xsd:string"/>
        <xsd:element name="ID" type="xsd:string"/>
        <xsd:element name="ProductType" type="xsd:string"/>
        <xsd:element name="Data">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name= "Desc"
                    type="xsd:string"/>
                    <xsd:element name="MfgDate"
                    type="xsd:date"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="ID" type="xsd:int" />
</xsd:complexType>

</xsd:complexType
name="LongNames_abcdefghijklmnopqrstuvwxyz_abcdefghijklmnopqrstuvwx
yz">
    <xsd:sequence>
        <xsd:element
name="LongElemName_abcdefghijklmnopqrstuvwxyz_abcdefghijklmnopqrstu
vwxyz_abcdefghijklmnopqrstuvwxyz_abcdefghijklmnopqrstuvwxyz"
type="xsd:string"/>
```

```
        </xsd:sequence>

      </xsd:complexType>

  </xsd:schema>
```

5   [0052]   When the conventions described above are applied to the snippet of
Table 1, the illustrative Java component names of FIG. 6 are generated (with
the user defined prefix being "XML"). The methodology by which Java
component names of FIG. 6 were preferably generated may be further
understood in the following (where the numbers in the square brackets

10   correspond to the row numbers in FIG. 6):

[0053] 1) Complex Type names to Java class names: Apply the hash code of the
QName (typesNS#Product) of the complex type. This guarantees uniqueness
of the Java class names generated in the system [1].

[0054] 2) Elements with same name and type: Take the hash code of the

15   element name followed by "Elem" (ID#Elem). If there are more than one such
element, add a suffix (0,1 etc.) in increments of 1 [2, 4, 6].

[0055] 3) Elements and Attributes with same name: For elements follow the
process in step 2. For attributes take the hash code of the attribute name
followed by "Attr" (ID#Attr) [6, 7].

20   [0056] 4) Anonymous Complex Types to Java class names: Create a string
comprised of the concatenation of all of the names of the components in the
hierarchy for the anonymous complex type starting from the schema element
including the target namespace of the schema. Generate a hash code using the
concatenated string. The name is generated by appending the hash code to the

25   name of the immediate parent component of the anonymous complex type.
"Data"'s type is anonymous. Its hash code is generated using
typesNS:Product#Data which is the path from the global type definition
(Product). This will guarantee uniqueness of Java class names among all
complex types (both named and anonymous) [8].

[0057] 5) Long Names (types, elements, attributes): Take the hash code of the entire name and truncate the name along with the hash code so that its length is at most the maximum size allowed for the name. In [11] the Java class name XML_N1640874745_LongNames_abcdefghijklmnopqrstuvwxyz_abcde"

5 is generated for the complex type name. This will enable uniqueness of names (both of Java class names and member variable names) as one takes the hash code of the entire name. This also helps adhere to the maximum name size constraint. A similar approach is used in mapping schema element and attribute names to Java member variable names [11, 12].

10 **[0058]** Of course, one skilled in the art will appreciate how a variety of alternatives are possible for the individual elements, and their arrangement, described above, while still falling within the scope of the invention. Thus, while it is important to note that the present invention has been described in the context of a fully functioning data processing system, those of ordinary

15 skill in the art will appreciate that the processes of the present invention are capable of being distributed in the form of a computer readable medium of instructions and a variety of forms and that the present invention applies equally regardless of the particular type of signal bearing media actually used to carry out the distribution. Examples of signal bearing media include

20 recordable-type media, such as a floppy disk, a hard disk drive, a RAM, CD-ROMs, DVD-ROMs, and transmission-type media, such as digital and analog communications links, wired or wireless communications links using transmission forms, such as, for example, radio frequency and light wave transmissions. The signal bearing media may take the form of coded formats

25 that are decoded for actual use in a particular data processing system. Moreover, while the depicted embodiment includes an example in a Java environment, the processes of the present invention may be applied to other programming languages and environments.

[0059] In conclusion, the above description has been presented for purposes of illustration and description of an embodiment of the invention, but is not intended to be exhaustive or limited to the form disclosed. This embodiment was chosen and described in order to explain the principles of the invention,

5   show its practical application, and to enable those of ordinary skill in the art to understand how to make and use the invention. Many modifications and variations will be apparent to those of ordinary skill in the art. Thus, it should be understood that the invention is not limited to the embodiments described above, but should be interpreted within the full spirit and scope of the

10   appended claims.